Democratizing Personalization

Antoine Boutet, Davide Frey, Anne-Marie Kermarrec, Rachid Guerraoui

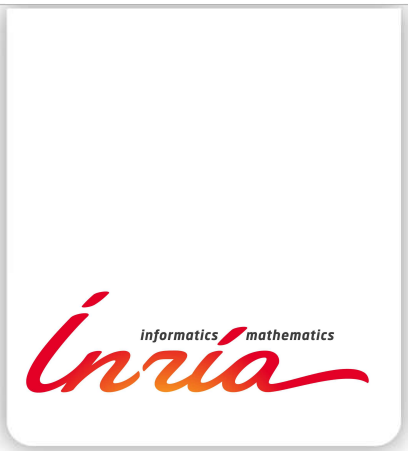
RESEARCH REPORT

N° 8254

February 2013

Project-Teams ASAP

ISSN 0249-6399 ISRN INRIA/RR--8254--FR+ENG



Democratizing Personalization

Antoine Boutet *, Davide Frey †, Anne-Marie Kermarrec ‡,
Rachid Guerraoui §

Équipes-Projets ASAP

Rapport de recherche n° 8254 — February 2013 — 20 pages

Résumé : The ever-growing amount of data available on the Internet can only be handled with appropriate personalization. One of the most popular ways to filter content matching users' interests is collaborative filtering (CF). Yet, CF systems are notoriously resource greedy. Their classical implementation schemes require a substantial increase in the size of the data centers hosting the underlying computations when the number of users and the volume of information to filter increase.

This paper explores a novel scheme and presents *DeRec*, an online cost-effective scalable architecture for CF personalization. In short, *DeRec* democratizes the recommendation process by enabling content-providers to offer personalized services to their users at a minimal investment cost. *DeRec* achieves this by combining the manageability of centralized solutions with the scalability of decentralization. *DeRec* relies on a hybrid architecture consisting of a lightweight back-end manager capable of offloading CPU-intensive recommendation tasks to front-end user browsers.

Our extensive evaluation of *DeRec* on reals workloads conveys its ability to drastically lower the operation costs of a recommendation system while preserving the quality of personalization compared to a classical approach. On average, over all our experiments, *DeRec* reduces the load on the back-end server by 72% when compared to a centralized alternative.

Mots-clés : recommendation systems, collaborative filtering

* antoine.boutet@inria.fr
† davide.frey@inria.fr
‡ anne-marie.kermarrec@inria.fr
§ rachid.gerraoui@epfl.ch

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Démocratisation des systèmes de personnalisation

Abstract: Un système de recommandation de news totalement distribué comporte plusieurs avantages du point de vue du passage à l'échelle et de la résilience aux pannes. Il permet aussi aux utilisateurs de librement l'exploiter sans aucune forme de contraintes liées à la rémunération du système, tel que la publicité par exemple. Cependant, les sites web commerciaux et les éditeurs de contenu sur le web n'ont pas encore trouvé de modèles économiques adaptés à cette architecture distribuée et préfèrent les systèmes centralisés pour facilement exploiter l'ensemble des données. La dernière contribution de cette thèse explore un nouveau modèle tirant parti des avantages des systèmes distribués tout en conservant une architecture centralisée. *DeRec* démocratise les systèmes de recommandations en offrant aux fournisseurs de contenu un système de personnalisation pour leurs utilisateurs, et ce à faible coût. *DeRec* fait appel à une architecture hybride composée d'un gestionnaire de tâches léger capable d'exporter la partie la plus importante du processus de recommandation sur le navigateur des utilisateurs. *DeRec* intègre également un mécanisme qui permet d'adapter les tâches effectuées par le serveur et par les clients en fonction de la charge du serveur et de la capacité de calcul du matériel de l'utilisateur. Notre solution est générique et peut facilement être adaptée à des systèmes existants.

Key-words: systèmes de recommandation, filtrage collaboratif

1 Introduction

Personalization is now essential to navigate the wealth of data available on the Internet. Recommendation systems are particularly popular and provide users with personalized content, based on their past behavior and on that of similar users. They have been successfully applied by on-line retailers such as Amazom.com, Facebook or Google or Yahoo ! News to suggest resp. items, friends or news. Yet, the need to personalize content is no longer an exclusive requirement of large companies. It is arguably crucial for every web content editor, including small ones. Most of these, besides publishing content (news, articles, reviews, etc), let users comment and discuss this content, generating of a continuous stream of information. Buying or renting the required computing power for an effective personalization represents a significant investment for small companies.

In this paper, we present and extensively evaluate *DeRec*, a novel architecture providing a cost-effective personalization platform to web content editors. Instead of scaling out recommendation back-ends, *DeRec* delegates expensive computation tasks to web browsers running on client machines, while, at the same time, retaining the system's coordination on the server side.

DeRec implements a *user-based collaborative filtering* (CF) scheme. CF [14] is the process of predicting the interests of a user by collecting preferences from other users. Its user-based variant is content agnostic and represents a natural opportunity for decentralizing recommendation tasks on users' machines, where each user is herself in charge of the computation of her personalization. *DeRec* adopts a k-nearest-neighbor strategy which computes the k nearest neighbors according to a similarity metric, and identifies the items to recommend from this set of neighbors [24]. The challenge is to cope with a large number of users and items. Traditional recommendation architectures achieve this by computing neighborhood information offline and exploiting elastic cloud platforms to massively parallelize the recommendation jobs on a large number of nodes [12, 13]. Yet, offline computation makes real-time recommendations very hard to achieve and forces the periodic re-computation of predictions, inducing significant running costs [12, 19] and power consumption [20].

DeRec's hybrid architecture avoids the need to process the entire sets of users and items at the same time by means of a sampling-based approach inspired from epidemic computing [25, 9]. The *DeRec* back-end server provides each front-end user's web browser with a sample set of other users. From this sample, the browser computes its user's k nearest neighbors and most popular items. The process continues with the server's using the user's new neighbors to compute the next sample. This leads to a feedback mechanism that improves the quality of the selected neighbors and leads them to converge very quickly to those that would have been computed using global knowledge. While this involves extra communication between the server and the clients, there is a clear advantage in terms of computational cost as demonstrated by our evaluations.

DeRec is generic and applicable to any CF system that processes the user-item matrix in a user-centric fashion. Content providers can customize *DeRec* with a specific sampling procedure or an application-tailored similarity metric. *DeRec* also incorporates an adaptation mechanism that can tune the computational tasks on both the server and the clients according to their capability footprints. *DeRec* can accommodate client machines ranging from large desktop computers to smaller mobile devices. It also allows the server to operate effectively even during load peaks.

The architecture of *DeRec* is hybrid in the sense that it lies between traditional centralized systems and fully decentralized solutions such as [23] and [21]. In this respect, *DeRec* provides the scalability of decentralization without forcing content providers to give up the control of the system. Unlike fully decentralized approaches, its lightweight web widget does not require clients to install specific software.

We extensively evaluate *DeRec* in the context of two use cases, a personalized feed, Digg, and

a movie recommender, MovieLens, using real traces in both cases. We compare *DeRec* against solutions based on a centralized infrastructure with an offline neighbor-selection process and an online item-recommendation step. Our results show that *DeRec* reduces the server's computational requirements by a factor ranging from 2.1 to 8.5 while preserving the quality of recommendation and with only limited computational and bandwidth costs for client machines. We show that, as the scale of the system increases, the load on the server grows much more slowly with *DeRec* than with a centralized approach. We also show that *DeRec* successfully adapts to various client capabilities by reducing the size of the sample provided by the server. This demonstrates the viability of *DeRec* as an alternative to data-center-based approaches by leveraging the computational power of clients. We believe that *DeRec* can be adopted by any content editor wishing to provide personalized recommendations to their users at a low cost.

2 Related Work

While content-based recommenders leverage the similarities between items, CF ones focus mainly on the users. Due to its content-agnostic nature, CF has now been adopted in a wide range of settings [14]. User-based [17] CF recommenders build neighborhoods of users based on their interest in similar (e.g. overlapping) sets of items. One of the main challenges underlying CF is scalability. This is particularly true for websites that enable users to generate content. Dimension reduction and algorithmic optimizations (*(e.g.)* [16, 15]) partially tackle the problem, but they do not remove the need to increase computational resources with the number of users and items [22, 12, 10]. Even with massive (map-reduce) parallelization [13] on elastic cloud architectures, CF systems's cost is high both in terms of hardware and energy consumption [1]. A more radical way to address scalability is through a significant departure from centralized (cloud-based) architectures, namely through fully distributed CF solutions [26, 23, 8, 21, 27]. Whereas elegant and appealing in theory, the requirement to have a software on every client, the need for synchronization between multiple devices, and the management of users' on/off-line patterns make these solutions hard to deploy. Yet, their inherent scalability provides a strong motivation for a hybrid approach like ours which separates the concerns : a centralized back-end handles the connections and disconnections of users whereas front-ends perform the actual personalization on the client side. A similar idea was partially explored by TiVo [7] in the context of an item-based CF system. It only offloads the computation of item-recommendation scores to clients, while it computes the correlations between items on the server side. Since the latter operation is extremely expensive, TiVo's server only computes new correlations every two weeks, while its clients identify new recommendations once a day. This makes TiVo unsuitable for dynamic websites dealing in real time with continuous streams of items. In contrast, *DeRec* addresses this limitation by delegating the entire filtering process to clients : it is to our knowledge the first system to do so and applicable to any user-based CF platform.

3 System model

User-Based CF. We consider a set of users $U = u_1, u_2, ..., u_N$ and a set of items $I = i_1, i_2, ..., i_M$, Each user $u \in U$ has a profile P_u collecting her opinions on the items she has been exposed to. This consists of a set of quadruplets $P_u = < u, i, v, t >$ where u is a user, i an item, and v the score value representing the opinion of user u on i as recorded at time t. For the sake

1. Data centers consume an enormous amount of power and energy requirements are responsible for a large fraction of their total cost of ownership and operation [11, 20].

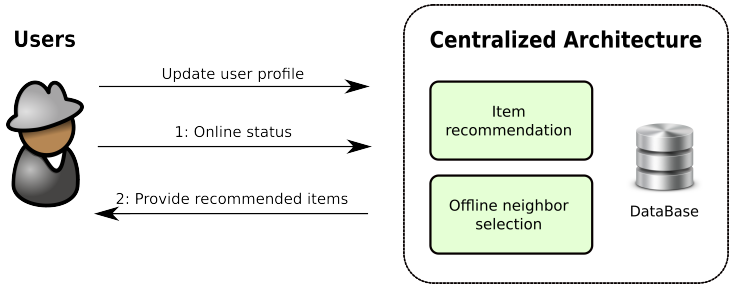


FIGURE 1: *Centralized Architecture (offline recommendation).*

of simplicity, we only consider binary ratings indicating whether a user *liked* or *disliked* an item after being exposed to it. [2]

The goal is to provide each user with a set of items $R \subseteq I$ which she is likely to appreciate. To achieve that, user-based CF systems operate in two steps : *neighbor selection* and *item recommendation*. Neighbor selection consists in computing, for each user u, the most similar users with respect to a given similarity metric. For the sake of simplicity, we consider here the well-known cosine similarity metric [14], computing the cosine of the angle formed by two users profiles $\sigma(P_u, P_n)$ where $u, n \in U$. [3] To avoid computing the similarity of a user with all other users, we consider a sampling-based approach to reduce the dimension of the problem. The CF system then uses the selected neighbors to recommend items to a user that she has not yet been exposed to. In this paper, we consider the top-r most popular items of the extended neighborhood as in [12].

Both the computation of neighborhood and item recommendation are solely based on the content of user profiles. In order to react immediately to new user requirements and potential changes of interests, recommendations take only into accounts the ratings done within a sliding time window (*profile window*). Depending on the dynamics of the application, the size of the profile window varies from a few hours to several months or may include only the x last ratings. Here, we consider only the opinions expressed by users on the last T_{win} items, where T_{win} is a system parameter. [4]

Centralized architecture. The typical architecture of a CF system for web applications follows a client-server interaction model (Figure 1). Users interact with a web browser, which in turn communicates with the web server of the application providing it with information about the user's interests (*e.g.* clicks or ratings) and receiving the results of the recommendation process.

In this paper, we consider the architecture described in [12] to compare *DeRec* to. The content provider stores the users' information on a database, and performs the tasks identified above. Due to its high computational cost, the neighbor selection is carried out offline, periodically, according to the dynamics of items. The item-recommendation task is performed in real time. The main characteristic of this architecture is that all computation-intensive tasks are performed at the server. The web server, application logic, recommendation subsystem, and associated databases may leverage distribution, for example using cloud-based solutions. Yet, all active components remain under the responsibility of the website owner. For instance, Google News [12] employs a cloud-based solution and delegates computation- and data-intensive recommendation tasks to a

2. This rating can be easily extended to multi ratings.
3. *DeRec* can easily be parametrized with other similarity metrics.
4. In CF systems, the presence of new users (*i.e* with empty profiles) and new items (*i.e.* without ratings) leads to the so-called *cold-start* issue. This, however, is application dependent. As our system can be easily parametrized to address this problem as needed, the cold-start evaluation is out of the scope of this paper.

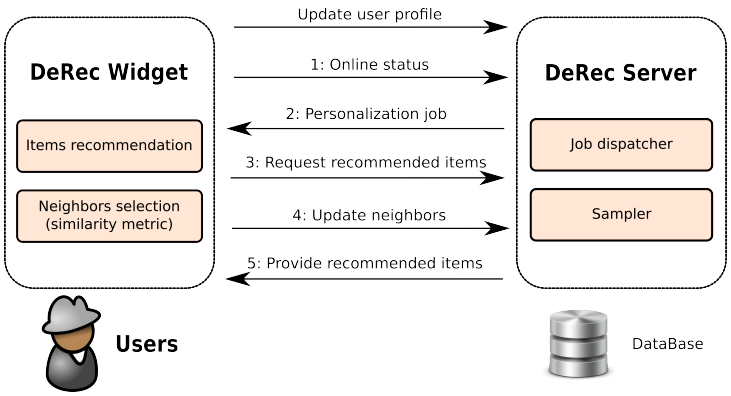


FIGURE 2: *Main tasks and components of* DeRec.

data center. This allows its recommendation system to scale to large numbers of users and items, but requires significant investments to provide recommendations within satisfactory response times. This makes such server-based architectures viable only for large companies that are able to sustain the associated costs.

4 DeRec

DeRec addresses the limitations of existing architectures by providing personalized recommendations without incurring the significant infrastructure costs associated with traditional recommendation solutions. *DeRec* achieves scalability by decentralizing the computation of the intensive tasks of the recommendation process on the browsers of online users, while maintaining a centralized system orchestration. This leverages connected users for scalability purposes in a transparent manner, without suffering from the limitations that characterize fully decentralized architectures.

The high-level architecture of *DeRec* is depicted in Figure 2. Similar to existing systems, users' web browsers interact with a front-end server that maintains the user-profile database and provides the recommended items once identified by the recommendation process. However, the computationally intensive tasks of item recommendation and neighbor selection are transparently offloaded to users and executed within their browsers. To achieve this goal, the *DeRec* server uses a sampling-based approach to efficiently split and dispatch personalization jobs to users. This approach limits network traffic by constraining the size of the set from which neighbors are selected while preserving the quality of the recommendation. The *DeRec* widget, executes neighbor selection and item recommendation through Javascript code and makes the decentralization transparent to users.

In the following, we first describe how the *DeRec* server distributes the computation of the two main steps of user-based CF. Then, we present the *DeRec* widget's operations and its interactions with the server, before introducing the footprint capability adaptation, allowing *DeRec* to dynamically adapt the size of the computation jobs sent to users to the capability of their devices and the load of the server.

4.1 *DeRec* server

The key feature of the *DeRec* server is the ability to decentralize the computationally intensive tasks of user-based CF. The recommendation cost is thus shared between the server and a large number of online users. The server directs personalization jobs (composed of item recommendation and neighbor selection) to online users by means of the two main components depicted in Figure 2 : the sampler and the job dispatcher.

In order to reduce traffic, *DeRec* does not require each user to select her k nearest neighbors from the entire database. Instead, the server uses the sampler to provide each user with a sample of candidates. This sampling-based approach allows *DeRec* to significantly reduce the size of the recommendation problem.

The job dispatcher packs each prepared sample into a personalization job (essentially a message containing the sample) to be executed by online users, and then collects the results of their computations. Consequently, the entire recommendation process in *DeRec* from the content provider's point of view is reduced to selecting a sample and to preparing, sending and collecting the results of the personalization job for the online users.

4.1.1 Sampler

The sampler is solicited by the job dispatcher to generate a sample of candidate users S with which a user computes her similarity. The sampler builds a sample $S_u(t)$ for u at time t by aggregating three sets : (i) the current neighbors N_u of u, (ii) their neighbors, and (iii) a random set of other users. Let k be a system parameter determining the size of a user's neighborhood, N_u. Then the sample contains at most k one-hop neighbors, k^2 two-hop neighbors, and k random users. Because these sets may contain duplicate entries, the size of the sample is $\leq 2k + k^2$. The job dispatcher can further reduce this size to take into account the capability footprints of both the user and the server (see Section 4.3).

The periodic computation of samples takes its inspiration from epidemic clustering protocols [25, 9]. Using u's neighbors and their neighbors provides the widget with a set of candidates that are likely to have a high similarity with u. Adding random users to the sample prevents this search from getting stuck into a local optimum and guarantees that the process will eventually converge by recording the user's k-nearest neighbors in the set N_u, so that $\lim_{t\to\infty} N_u - N_u^* = 0$, where N_u^* is the optimal set (*i.e.* containing the k most similar users). Research on epidemic protocols [25] has shown that convergence is achieved very rapidly even in very large networks.

4.1.2 Job dispatcher

The job dispatcher manages the distribution of the personalization jobs to users. Once it receives an online notification from a user u (arrow 1 in Figure 2), it asks the sampler for the subset of candidates described above, the sample. In doing so, it asks the sampler for an adjusted sample size according to the capability footprints as explained in Section 4.3. Then, it prepares the personalization job for u by building a message including its profile and the profiles of all candidates returned by the sampler (arrow 2 in Figure 2). Finally, it manages the interaction with the *DeRec* widget : sends the personalization job and collects the results of neighbor selection and item recommendation. First, it stores the former into the database. Then, it processes the latter by sending the widget the actual content of the selected items (arrow 5 in Figure 2).

4.2 *DeRec* widget

Similar to existing systems, users interact with *DeRec* through a web interface that provides them with personalized feeds of items. In *DeRec*, this is a widget written in Javascript, acting as web container and interacting with the *DeRec* server using a web API. Javascript has been widely adopted and makes it possible to create dynamic web interfaces, for example by proactively refreshing their content. *DeRec*'s widget leverages this technology to massively distribute the main tasks of recommendation to the browsers of online users.

The *DeRec* widget sits on the client side and manages all the interactions with the server side. Consider a user u. First, the widget is responsible for updating the user profile stored on the server. To achieve this, it contacts the server whenever u expresses an opinion on an item (*update-profile* arrow in Figure 2). Second, the widget is responsible for refreshing the displayed recommendations. To achieve this, it periodically contacts the server with an *online-status* message indicating that the user is online (Arrow 1 in Figure 2). The server replies to this message by sending a personalization job containing a sample of users along with their associated profiles (Arrow 2). Upon receiving this job, the widget computes u's personalized recommendations as $R_u = \alpha(S_u, P_u)$, where $\alpha(S_u, P_u)$ returns the identifiers of the r most popular items among those that appear in the profiles in S_u, but not in P_u. These are the most popular items in the sample to which u has not yet been exposed. The widget then requests the actual content of these items by sending the selected identifiers to the server (Arrow 3 in Figure 2). When the server replies to this request (Arrow 5), the widget displays the items to the user.

It is worth remembering that the sample, S_u, contains mostly users that are in u's two-hop neighborhood, together with a small number of randomly selected users. By taking into account the items appreciated by the former, the widget exploits the opinions of similar users. By also taking into account those appreciated by the latter, it also includes some popular items that may improve the serendipity of its recommendations.

After requesting the recommended items from the server, the widget also proceeds to updating the user's k-nearest neighbors. To achieve this, it computes the similarity between u's profile and each of the profiles of the users in the sample. It then retains the users that exhibit the highest similarity values as u's new neighbors, $N_u = \gamma(P_u, S_u)$, where $\gamma(P_u, S_u)$ denotes the k users from S_u whose profiles are most similar to P_u. It then returns these neighbors to the server to update the database (arrow 4 in Figure 2). The server will use these new neighbors the next time it has to compute a new sample.

4.3 *DeRec* capability footprint adaptation

To account for the heterogeneity of user devices, *DeRec* provides a device-capability adaptation mechanism. More specifically, the widget periodically computes a capability footprint of its device and sends it to the server. This footprint measures the time spent by the device to solve a computational puzzle (*i.e.* reversing an MD5). The *DeRec* server keeps track of this capability footprint of each online user. Since, there is a direct correlation between the size of the sample and the cost of the personalization job, the *job dispatcher* uses the footprint capability to adapt the size of the *sample* send to a user. To this end, content providers can define a function that maps each capability-footprint value onto a percentage p_{cap}. The job dispatcher then asks the sampler for a sample consisting of $kp_{\text{cap}}/100$ one-hop neighbors, $(kp_{\text{cap}}/100)^2$ two-hop neighbors, and $kp_{\text{cap}}/100$ random nodes.

Similarly, the server might also experience picks in load that limit its operation. This is reflected in the server capability footprint, directly correlated to the number of connected users. *DeRec* also accounts for such limitations by adapting the server's operations according to its

footprint. Similarly to the clients, the server uses an internal footprint to adapt the size of the samples that constitutes candidate list. As for the clients, the adapted size is expressed as a percentage of the default sample, parametrized by k.

5 Implementation and Evaluation

5.1 Implementation

Our implementation of *DeRec* consists of a set of server-side modules and a client-side widget as described in Section 4. Server's component are J2EE servlets, either bundled all together with a version of Jetty [3], a lightweight web-server, or as stand-alone components that can be run in a web server. Integrating all components into a customized web server allows content providers to deploy our solution into their existing web architectures. Moreover, bundling each component with a Jetty instance makes it easy to deploy our architecture on multiple hosts thereby balancing the load associated with the various recommendation tasks (*e.g.* network load balancing).

The *DeRec* client consists of a web widget, a piece of Javascript acting as a web container that can easily be embedded in third-party sites, or online and offline dashboards (*i.e.* netvibes, igoogle, web interface). This Javascript defines the behavior of the widget : it collects user opinions, executes the personalization jobs, receives recommendations, and displays them. To do so, it communicates with the *DeRec* server through a web API not described here for space reason. The use of a public web API not only provides a simple way to implement our widget but also achieves authentication and makes it possible for content providers and even users to build their own widgets that interact with the framework. To develop a new widget, one simply needs to make the right API calls and import the Javascript file associated with *DeRec* widget that deals with processing selection jobs. All exchanges from the server to the widgets of online users are formatted in JSON. We use the Jackson implementation [2], one of the fastest solutions to serialize JAVA objects to JSON message. To parameterize *DeRec*, content providers can specify a specific similarity metric or item recommendation algorithm in the Javascript file which defines the behavior of the widget, and includes the desired fields of the user profile in the JSON messages. The current version of *DeRec* integrates interfaces to easily customize parts of its behavior.

5.2 Experimental setup

5.2.1 Platform

In our experiments, we consider a single server hosting all components (front and back-end) and we assume that the database is entirely stored in memory. Obviously in practice, several machines can be used to implement each component separately to sustain the load at the network level. However, as this load balancing technique does not affect the outcome of our experiments, its evaluation is out of scope in this paper. In our experiments, we use two PowerEdges 2950 III, Bi Quad Core 2.5GHz, with 32 GB of memory and Gigabit Ethernet, respectively to evaluate the server and the clients.

5.2.2 Datasets

We use real traces from a movie recommender based on the MovieLens workload [6] and Digg [1], a social news website. The MovieLens dataset consists of movie-rating data collected through the MovieLens recommender web site during a 7-month period. For the sake of simplicity, we project the ratings into a binary rating as follows : for each item in a user profile, the rating

is set to 1 if the initial rating of the user for that movie is above the average rating of the users across all her items, 0 otherwise. We use three sizes of this dataset to assess how *DeRec* scales up when the number of users increases.

Dataset	Users	Items	Ratings
MovieLens1 (ML1)	943	1,700 movies	100,000
MovieLens2 (ML2)	6,040	4,000 movies	1,000,000
MovieLens3 (ML3)	69,878	10,000 movies	10,000,000
Digg	59,167	7,724 items	782,807

TABLE 1: *Datasets statistics*

We also use a Digg dataset to study a situation with a highly dynamic feed of items. Digg is a social news website to discover and share content where the value of a piece of news is collectively determined. We collected traces from Digg for approximately $60,000$ users and more than $7,500$ news over 2 weeks in 2010. This dataset contains all observed users in the specified period. Table 1 summarizes the workload figures.

5.2.3 Online patterns

In *DeRec*, a recommendation is computed for a user when she is online. In order to evaluate the impact of the number of online users on *DeRec*, we consider several online patterns. To this end, we use the timestamp attached to user ratings in the datasets. We split the trace in timeslots and select the online users for each slot. A user is considered online if she provided at least one rating during a slot. Moreover, as users can be connected to *DeRec* without providing ratings (*i.e.* reading recommendations), we artificially add online users. We define an online pattern as a percentage of additional random users added to the ones which provided rating at each time slot. An online pattern of 0% means that only users which provided a rating at the associated time slot are considered.

5.2.4 Methodology

In order to evaluate *DeRec*, we run experiments simulating its operations over time by replaying the activity and ratings of users. In each slot, every online user sends an online notification to the server and in turn receives a sample from the *DeRec* server and uses it to perform the personalization tasks. Upon completion of those tasks, each client *(i)* sends a request to the server to get the recommended items and *(ii)* send an update of its k nearest neighbors. The server then provides the desired items to users and updates the profile database. To simulate real exchanges, each item provided by the server contains real content from a RSS feed item of 1004 bytes. Users also send their profile updates to inform the server about the items they liked or disliked for each rating present in the dataset during the associated slot. As described in Section 4.3, the job dispatcher tunes the size of requested samples according to footprint capabilities. To model this, as well as the mapping of capabilities onto percentages we associate each user with a random value between 40 and 100. This represents the percentage of the neighborhood size, k, taken into account by the sampler. For simplicity, we abuse terminology and refer to this value as capability footprint. A user u with a 50% footprint in a system with $k = 10$, will receive a sample containing its 5 closest neighbors, the 5 closest neighbors of each such neighbor, and 5 random users, thus receiving a sample of size 35. Unless specified otherwise, we set the capability footprint for the

DeRec server to 100%. Finally, users send a logout notification at the end of the experiment. Default parameter values used in our experiments are summarized in Table 2.

Parameter	Value
Size of the neighborhood	10
Sample size	≤ 120
Time slot (MovieLens)	12 hours
Time slot (Digg)	1 hour
Recommended items	10 items
Online pattern	5%
Server capability footprint (server)	100%
Device capability footprint (clients)	[40-100%]
Windows profile	100 items
Offline clustering period (MovieLens)	48 hours
Offline clustering period (Digg)	24 hours

TABLE 2: *Default parameter setting*

5.2.5 Metrics

We measure the time spent on both the *DeRec* server and the *DeRec* widget. The reported time includes the time needed to receive and send the packets from or to the server and the widgets. In addition, we measure the bandwidth consumption between the *DeRec* server and the *DeRec* widgets. Regarding the recommendation quality, similar to most machine learning evaluation methodologies, we split the datasets in sub training and test sets (80% training set - 20% testing set). We report on both the precision and recall for users according to the number of items provided to users. Precision is defined as the number of interesting items received over the total number of received items. On the other hand, the recall is the number of interested items received over the number of items which should be received. To compare our solution with previous works [12], we do not use windows profile and consider all users ratings for the evaluation of the recommendation. Similarly, for the sake of comparison and to standardize the results, we mainly report them on the smallest MovieLens dataset (*i.e.* ML1).

5.3 Baseline for comparison

We compare *DeRec* against the centralized recommender solution depicted in Figure 1. In order to ensure a fair comparison, we select several alternative approaches and use the least expensive as a baseline in the rest of the experiments. The alternative approaches differ only in their k-neighbor selection algorithm, and use the same client-server protocol as *DeRec*.

5.3.1 k-neighbor selection

In a centralized architecture, the neighbor selection task is achieved periodically offline by the server. More precisely, we define an offline clustering period of 48 hours and 24 hours for the MovieLens and Digg datasets respectively, as reported in Table 2. We consider several alternatives to select the k-nearest neighbors of each user : *(i)* an exhaustive and multi-threaded approach computing the similarity between a user and all other users in the system ; *(ii)* a solution provided by Mahout, an open-source machine-learning Apache library [5] ; *(iii)* a probabilistic approach using Minhash similar to [12] available in [4], relying on a clustering model which performs

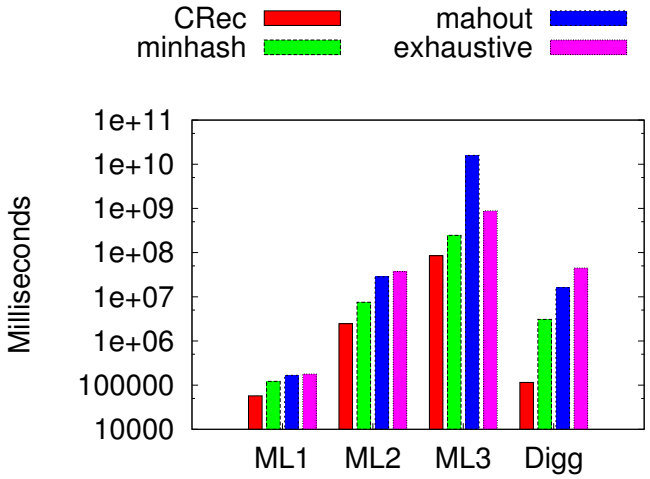


FIGURE 3: *Time to compute the k nearest neighbors.*

dimension reduction of the problem by assigning users to the same cluster according to the probability of the overlap between the set of items they liked in common ; *(iv)* a multi-threaded solution, called *CRec*, using the same algorithm as *DeRec* (*i.e.* using a sampling approach) but executed in a centralized manner.

We evaluate the total amount of time spent to achieve the k-neighbor selection task on all centralized candidates. Their parameters are set to the same values as in *DeRec*. More precisely the profile window is limited to the last 100 rated items. To simulate a parallel computation on a 10 node cluster of both the Mahout and Minhash approaches, we report a lower bound of the computation time *(i.e.* divided by 10). The results are depicted in Figure 3. We observe that *CRec* consistently outperforms other approaches. Therefore, we select *CRec* to compare *DeRec* with in the rest of this paper. Note that the quality of the recommendation provided by all these approaches is similar (as shown in Section 5.9).

5.3.2 Item recommendation

In the centralized architecture, the same recommendation as for *DeRec* is used : every time a user sends an online status to get a recommendation, the server computes the most popular items among her extended neighborhood (*i.e.* composed of its direct neighbors and their neighbors and random users) and then provides to the client the r most popular items unknown to the user.

5.4 *DeRec* versus *CRec* (representative of the centralized approach)

5.4.1 Server evaluation

Table 3 presents the breakdown of computation times among the different operations managed by the *DeRec* server for the ML1 MovieLens dataset. Results show that the most time-consuming task in *DeRec* server is the one forming and managing the JSON messages to send the samples to users. Other operations such as the sampling itself, or the management of neighborhood and profile updates, recommendation requests, and logout notifications are negligible.

Figure 4 and Table 4 depict the comparison of *DeRec* against *CRec*. We show that the total amount of time consumed on the server is drastically reduced in *DeRec* compared to *CRec*.

Task	Time (ms)	%
Profile updates	1,297	2.6
Logout	12	0.02
Sampling	1,226	2.4
Build the sample message	38,200	**76.8**
Neighborhood updates	842	1.6
Recommendation requests	4,259	8.5
Build the recommended items message	3,874	7.7
Total	49,710	100

TABLE 3: DeRec *server operations for ML1.*

Indeed, in *DeRec* the message management dominates over computing time and consumes much less time, on average 25% for the neighbor selection on MovieLens datasets, and 92% for the item recommendation on all datasets. On the Digg dataset, *DeRec* spends more time for the neighbor selection task than *CRec* for the offline clustering. This is mainly due to the small size of the users profiles on the Digg dataset (on average 13 for Digg versus compared to 138 for MovieLens). Indeed, the computation time of offline clustering depends on the size of user profiles, in contrast to *DeRec* in which this is restricted to message management. In addition, the cost of offline clustering is correlated with its frequency : the more frequent it is, the longer it requires. In ML1, if we consider *CRec* with an offline clustering period that is twice as fast (*i.e.* 24 hours), *DeRec*'s improvement increases from 30% to 65%.

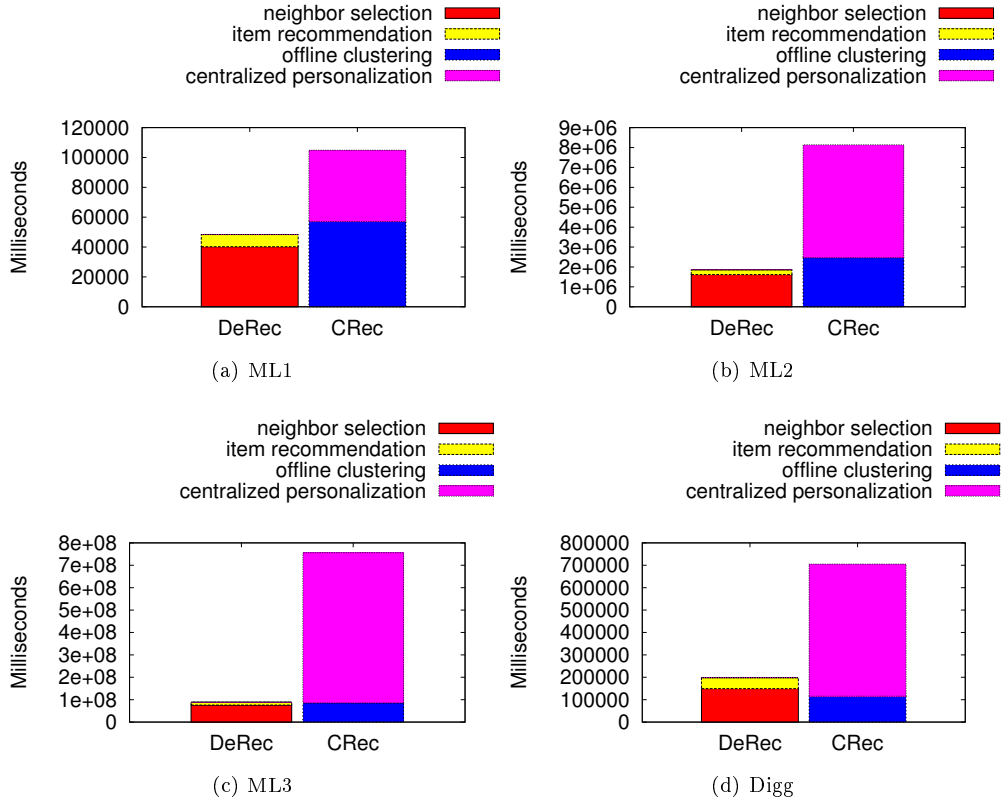
Component	ML1	ML2	ML3
DeRec server	49,710	1,859,213	88,609,095
All *DeRec* widgets	219,642	7,092,467	336,504,783
CRec	104,812	8,138,192	756,289,396

TABLE 4: DeRec *server versus* CRec *(ML1).*

Moreover, the gap between *DeRec* and *CRec* increases according to the number of users. The server in *DeRec* sees its load increase twice more slowly than *CRec* with a 6-fold scale increase in the number of users (*i.e.* ML1 to ML2), and four times more slowly with a 70-fold scale increase (*i.e.* ML1 to ML3). While the total computation time in all *DeRec* widgets is larger than the time spent by *CRec* for the entire recommendation process of ML1, from a content provider perspective, the load on its infrastructure is reduced by a factor of 2.1.

5.4.2 *DeRec* widget evaluation

We now evaluate the cost of operating *DeRec* on the client. The new action introduced on the client by our solution compared to the centralized one, is the management of personalization jobs including the item recommendation, the k-nearest-neighbor computation and the update messages sent to the server. We measure the time needed by the widget to achieve these different tasks on Table 5. We observe that about 50% of the time is spent on the item recommendation and the neighbor selection, the other 50% being shared by the request and the reception of the recommended items, the profile and the neighborhood updates. Similar to the *DeRec* server, the message management dominates over computing time even on the *DeRec* widget which limits the computation capability required on the client.



(a) ML1 (b) ML2

(c) ML3 (d) Digg

FIGURE 4: DeRec *server versus* CRec *(all datasets).*

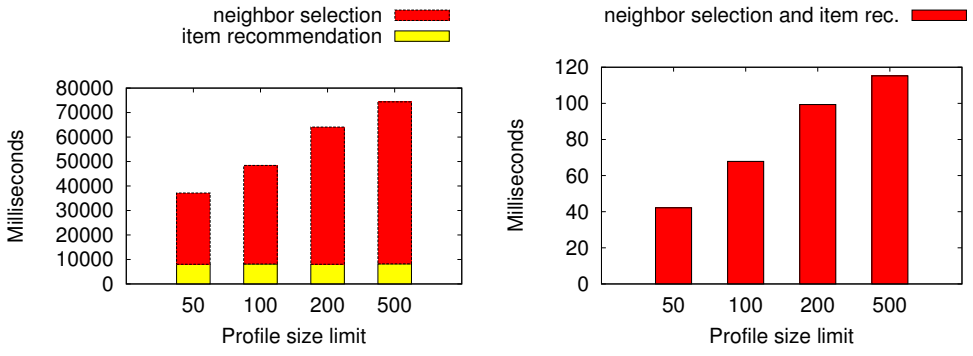
Finally, we measure the time spent by the widget within a browser (*i.e.* Firefox) to perform the selection jobs. An average of 911 milliseconds is spent by the browser to select the k nearest neighbors from the received sample and to identify the most popular items in its extended neighborhood. This task takes only 5.6 times longer than the time required by the widget to get and display an RSS feed from Digg, which makes *DeRec* clearly acceptable for users. In addition, this task is entirely transparent to them thanks to the asynchronous communication of the AJAX model.

Task	Time (ms)	%
Profile updates	22.16	17.2
Logout	0.19	0.1
Neighborhood updates	10.79	8.4
Neighbor selection & items recommendation	67.90	**52.9**
Requests and receives desired items	27.12	**21.1**
Total	128.16	100

TABLE 5: DeRec *widget operations for ML1.*

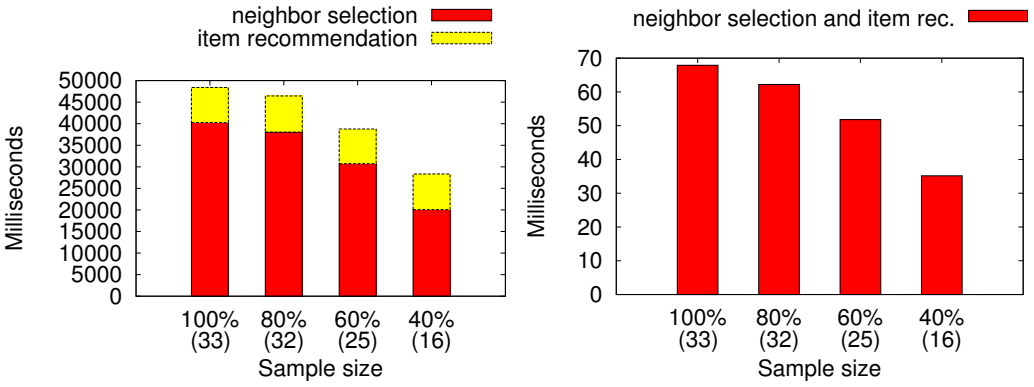
5.5 Impact of the profile-window size

The size of the user profile directly impacts the performance of *DeRec*. This largely depends on the dynamics of applications : typically users tend to rate much more often news than movies. More precisely, the larger the profile, the larger the size of the JSON messages generated by the server. The windows size directly impacts *DeRec* since it increases or decreases the time spent by the *DeRec* server to build up JSON messages. Figure 5 shows that the time required to prepare selection jobs increases only by a factor of 2 when the profile size changes from 50 to 500. Obviously, the time spent by the server to provide recommendation does not change according to the profile window (5a). However, on the *DeRec* widget, the time increases according to the size of the profile (5b).



(a) *DeRec* Server (b) *DeRec* Widget

FIGURE 5: *Impact of the profile window size for ML1.*



(a) *DeRec* Server (b) *DeRec* Widget

FIGURE 6: *Impact of the capability footprint for ML1.*

5.6 Capability footprint adaptation

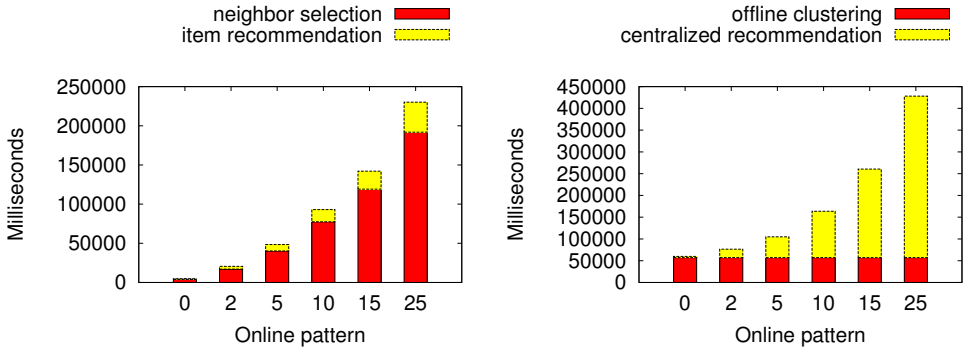
We analyze *DeRec*'s ability to account for the heterogeneous capabilities of user devices and balance the load on the server at the application level. We measure the time spent to achieve the different actions on both the server and a widget according to its load and its device capability, respectively. Figure 6 shows that by adapting the size of the sample sent to the widget, the server can reduce the time spent to form the sample message up to 45% for a server capability footprint of 100% vs 40%. On the other hand, the widget can reduce the required computation up to 48.2% from a device capability footprint from 100% to 40%.

Interestingly enough, due to the homophily characteristic of social networks [18], the average sample size is notably smaller than the maximum possible sample size (*i.e.* assuming that all neighbors of neighbors are distinct) Without any adaptation, the average sample size is equal to 33 against an upper bound of 120 (*i.e.* sum of the 10 random users and 10 neighbors of every of the direct 10 neighbors).

5.7 Online patterns

The number of users online directly impacts the load of the server. Figure 7 compares *DeRec* with a centralized architecture when facing a growing number of online users. In *DeRec* (7a), the main impact of a growing number of connected users is to generate more jobs to form and to send to online users. In contrast, in the centralized architecture (7b), the offline clustering takes the same time regardless of the number of clients. However, the time spent to achieve the item recommendation grows exponentially according to the online pattern as the server computes item recommendation for all online users.

However, unlike the *DeRec* server, the *DeRec* widget is not impacted by a growing number of users : each user is in charge of her own computation on a given sample provided by the server and the size of which does not vary with the size of the system. Figure 8 shows that for an online pattern from 2 to 25, the average time spent by users remains around 4 milliseconds for the MovieLens datasets and around 0 for the Digg one. This time varies according to the average size of the users profiles, the larger the profiles, the longer the computation time. An online pattern of 0 means that only users providing ratings in a time slot were defined as online in this slot, in contrast to a positive online pattern which includes additional random nodes as



(a) *DeRec* (b) *CRec*

FIGURE 7: *Server with varying online patterns for ML1.*

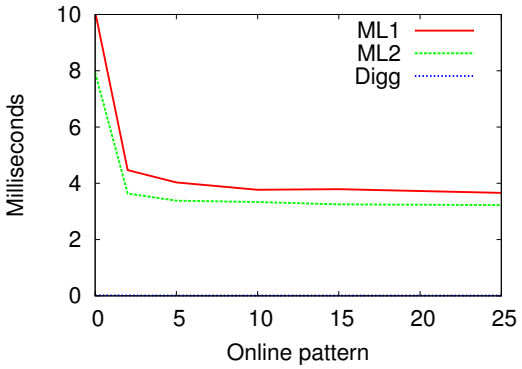


FIGURE 8: DeRec *widget with varying online patterns.*

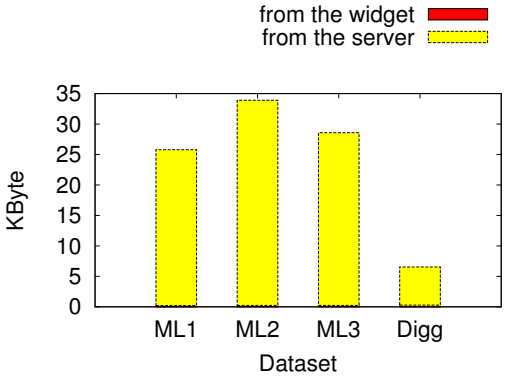
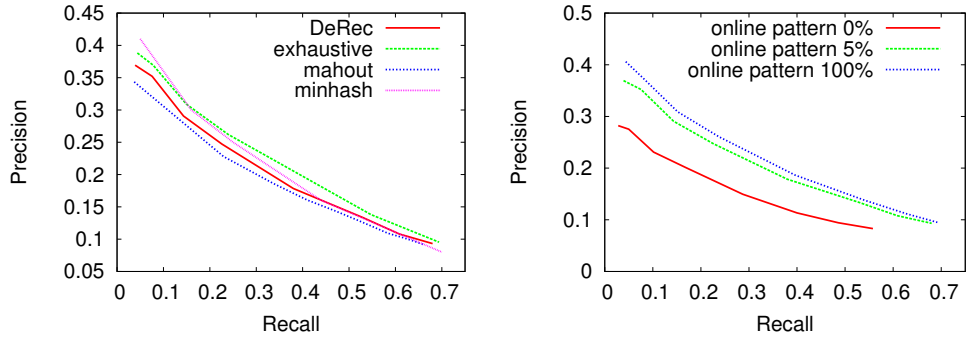


FIGURE 9: *Communication overhead of* DeRec *(ML1).*

online to generate artificially more activity. It is well known that in social platforms, the most active users in term of rating are connected more often than the others. As a consequence, in this case, online users are mainly the ones with the largest profiles and requires more time to compute the similarity. This explains the gap between online patterns of 0 and other values on Figure 8.

5.8 Communication overhead

By delegating expensive computation tasks to clients, the *DeRec* server and widgets experience a communication overhead compared to a centralized architecture. Figure 9 shows the average overhead generated by the server and the widgets according to the dataset, each time a user sends an online status to get recommended items. Results show that most of the overhead is generated by the server due to the sample message carrying user identifiers and their associated profiles. The clients only generate little overhead due to the notifications returned to the server when they have finished their tasks. However, this overhead is negligible compared to the average size of a Facebook page of 160.3 KBytes.



(a) According to approaches (b) Impact of the online pattern

FIGURE 10: *Recommendation analysis for ML1.*

5.9 Recommendation quality

We now evaluate the quality of recommendation to show that it is not impacted by *DeRec*. First, we evaluate the precision-recall trade-off achieved by *DeRec* compared to the other centralized candidates. Figure 10a shows that the quality of the recommendation provided by *DeRec* is similar to Mahout, and the Minhash solution as presented in in [12]. As in *DeRec*, the k nearest neighbor selection is refined only when users are online, we evaluate the impact of the online pattern on recommendation. As shown on Figure 10b, the more the available users, the higher their chance to benefit from recommendation. However, only a small online pattern is needed to provide good recommendations : an online pattern of 5% gives about 95% of the maximum precision-recall. This is is mainly due to the fact that neighbor selection is almost as good when computed on a sample as on the entire dataset. Although not reported here for space reason, we observe that the size of the neighborhood does not impact significantly the quality of the recommendation.

6 Concluding remarks

In this paper explores a novel hybrid architecture for personalizing recommendations by combining the scalability and cost-effectiveness of massively distributed systems with the ease of management of centralized solutions. We convey the feasibility of the approach through *DeRec*, a generic user-based collaborative filtering system that can be adopted by various web applications. As shown by our exhaustive experiments on several datasets, *DeRec* indeed constitutes a viable and promising alternative to data-center based approaches by leveraging the hardware and computation power of client machines. Content providers can drastically reduce their cost by reducing the required resources dedicated to a personalized recommendation system. *DeRec* is generic and could be adapted to many contexts. Exploring other similarity metric and recommendation algorithms represent an interesting perspective. *DeRec* currently exposes a users profiles, current work includes hiding the association user/profile.

Références

[1] Digg. `http://digg.com`.

[2] Jackson. `http://jackson.codehaus.org/`.

[3] Jetty. `http://www.eclipse.org/jetty`.

[4] Likelike. `http://code.google.com/p/likelike`.

[5] Mahout. `http://mahout.apache.org`.

[6] Movielens. `http://www.grouplens.org/node/73`.

[7] K. Ali and W. van Stam. Tivo : making show recommendations using a distributed collaborative filtering architecture. In *KDD*, 2004.

[8] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. A multi-agent infrastructure for developing personalized web-based systems. *ACM TOIT*, 2005.

[9] M. Bertier, D. Frey, R. Guerraoui, A.M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware*, 2010.

[10] M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM ICDM*, 2003.

[11] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *EuroSys*, 2012.

[12] A.S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization : scalable online collaborative filtering. In *WWW*, 2007.

[13] J. Dean and S. Ghemawat. Mapreduce : simplified data processing on large clusters. *Commun. ACM*, 2008.

[14] M.D. Ekstrand, J.T. Riedl, and J.A. Konstan. *Collaborative Filtering Recommender Systems*. Now Publishers, 2011.

[15] R. Gemulla, E. Nijkamp, P.J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, 2011.

[16] R. Grover and M. J. Carey. Extending map-reduce for efficient predicate-based sampling. In *ICDE*, 2012.

[17] J.L. Herlocker, J.A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR*, 1999.

[18] H.W. Lauw, J.C. Shafer, R. Agrawal, and A. Ntoulas. Homophily in the digital world : A livejournal case study. *IEEE Internet Computing*, 2010.

[19] G. Linden, B. Smith, and J. York. Amazon.com recommendations : item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.

[20] D. Meisner, C.M. Sadler, L.A. Barroso, W.D. Weber, and T.F. Wenisch. Power management of online data-intensive services. *SIGARCH Comput. Archit. News*, 2011.

[21] B. N. Miller, J.A. Konstan, and J. Riedl. Pocketlens : toward a personal recommender system. *ACM TOIS*, 2004.

[22] Jia P. and Dinesh M. Bi-level locality sensitive hashing for k-nearest neighbor computation. In *ICDE*, 2012.

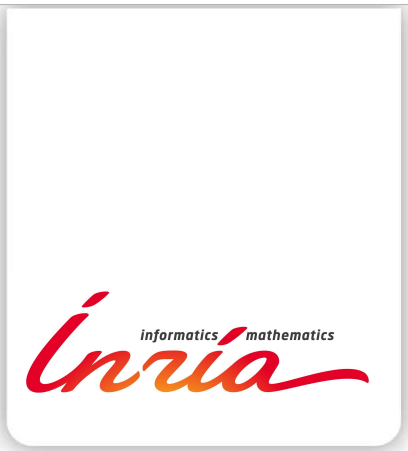
[23] D. Rosaci, G.M.L. Sarné, and S. Garruzzo. Muaddib : A distributed recommender system supporting device adaptivity. *ACM TOIS*, 2009.

[24] X. Su and T.M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009.

[25] S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par*, 2005.

[26] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse. Tribler : P2p media search and sharing. In *MM*, 2011.

[27] S. Zhang, G. Wu, G. Chen, and L. Xu. On building and updating distributed lsi for p2p systems. In *ISPA*, 2005.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399